Final Report

Names: James Fesik, Natalie Nguyen, Jorge Otero, Nicholai Platonoff, Michele Stewart

Abstract

The Litter Eliminator Pro (LEP) is a robot that autonomously navigates its environment and recognizes whenever a piece of trash enters its field of vision. Once it recognizes this trash, it will use a retrieval method similar to a brush and dustpan to pick up the trash, and a ramp that behaves as the dustpan will be lifted up to have the trash slide into the container used for storing trash. Trash detection is accomplished via a neural network model coded in Python. Autonomous navigation is achieved using simultaneous localization and mapping (SLAM). An Intel RealSense camera will be used to provide live images to the ML model. A laptop sitting wirelessly on the chassis processes the ML model, SLAM, and runs the ROS architecture. The output of the ML model and SLAM together is sent to the Teensy 4.1 microcontroller . The Teensy uses these signals to move the DC motors and servo motors. For the DC motors, the Teensy must pass the signal to an Adafruit Featherwing over I2C. The servo motors are directly controlled by a PWM pin from the Teensy.

Introduction

Litter Eliminator's goal is to solve the problem of trash pollution in urban areas. Our team's plan is to design a robot that will autonomously move around an area, detect any trash, and dispose of it properly. This project's intention is to create a positive environmental impact in the public spaces that we all share by reducing the amount of pollution. It is important to keep our public spaces clean for the environmental impact and the impact it has on the people who share these spaces. This autonomous garbage collecting robot could offload ground keepers from tedious and time demanding work, allowing them to focus on other important tasks.

Background

We draw inspiration from prior works that shed light on innovative mechanisms for waste collection. Firstly, the research by Jinqiang Bai and his team [4] utilizes deep learning algorithms, path-finding techniques, and a manipulative arm to gather discarded items from grassy terrain. We diverge by not using an arm and instead use a sliding and lifting mechanism to pick up trash, resulting in improved garbage collection efficiency, as shown by our comparative testing against other methods.

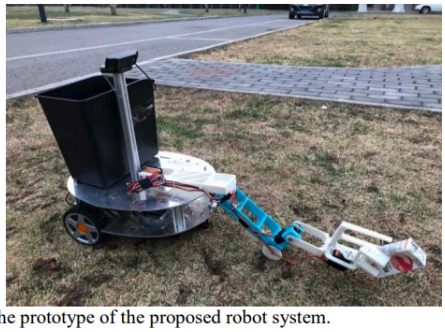

Fig. 1. The prototype of the proposed robot system.

Furthermore, another notable venture in the domain of waste management is the DustClean project [5], a model for street cleaning in Italy. This robotic system employs dual

brushes to corral litter toward its "mouth." It includes the Robot Operating System (ROS) and obstacle avoidance systems to navigate predefined routes and cleanse designated areas. Our approach differs in that our robot detects and retrieves litter proactively from its surroundings, focusing on the identification and retrieval of trash to accelerate and streamline the cleaning process.

ML:

“As a field of study, machine learning sits at the crossroads of computer science, statistics and a variety of other disciplines concerned with … inference and decision-making under uncertainty” [1].

Basically, machine learning seeks to find a mapper function from given inputs to desired outputs. The best mapper function tries to find the minimum error between all inputs X and corresponding outputs Y. Technically, finding a regression line (line of best fit) could be considered a machine learning task. Below is a plot from my Fundamentals of ML notes.

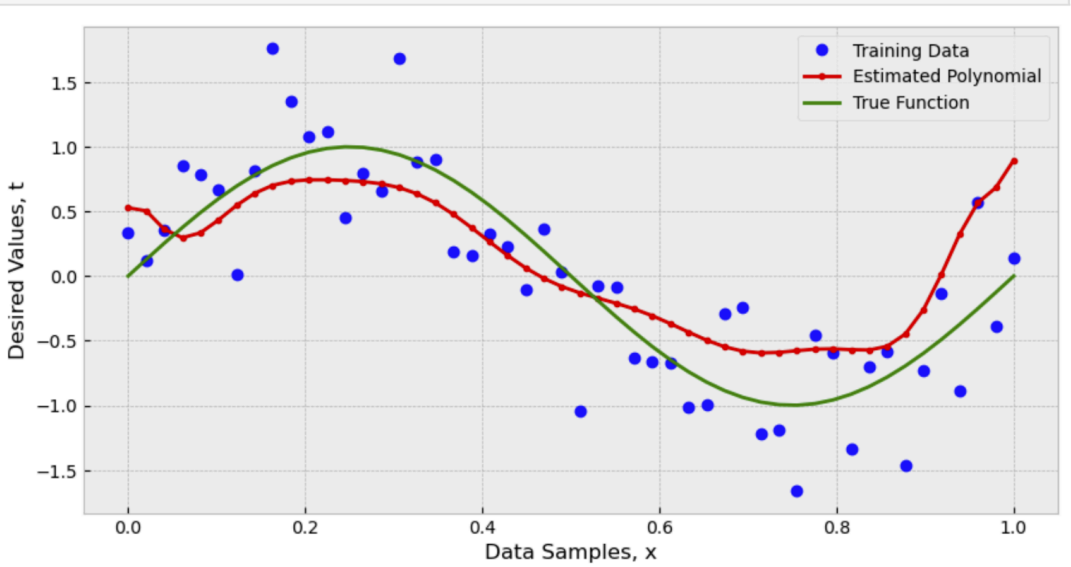

Figure 1: Simple Regression ML Task

The “training data” is created by sampling the “True Function” at some random variance away from it. The mapper function was a sum of polynomials(red) trying to recreate the sinusoid. The distance from all points to the true sinusoid was minimized. However, this is not the only solution; Hyperparameters like the degree of the polynomials and number of polynomials used will affect the outcome. Below is an example illustrating this.

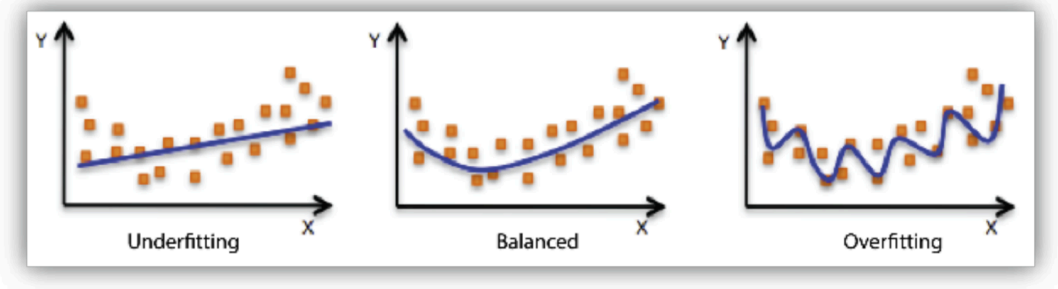

Figure 2: Possible Outcomes of Training

The created mapper function may either be too complex or not complex enough. In more complex scenarios like Object Detection for our project, many experiments and strategies need to be employed to create a balanced model. In general, more training photos with variable environments is the best action.

Object Detection still uses the same fundamentals principles discussed above. However, the mapper function cannot be visualized in a 2-D representation: it is multidimensional based on the pixel data. Also, polynomial mapping is not used for Object Detection or any other complex ML tasks:

"Many different forms of mapping *f* exist, including decision trees, decision forests, logistic regression, support vector machines, neural networks, kernel machines, and Bayesian classifiers" [1].

For my project, I have used Support Vector Machines(SVM) and Neural Networks. While not very similar, both of these mapper types try to extract and separate meaningful "features" in the data. Below is an example of how a neural network, specifically with convolutional layers (CNN), works to identify features.

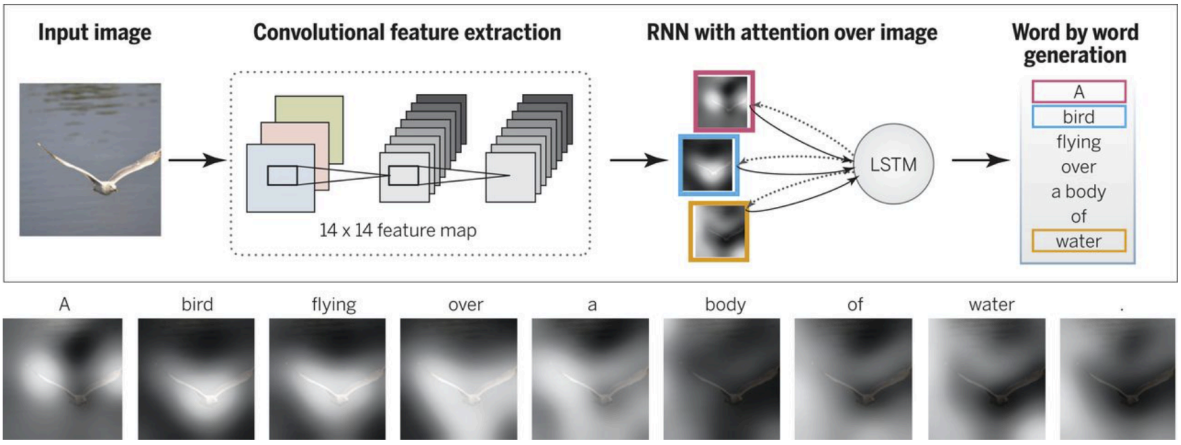

Figure 3: Convolutional Network Fed to Language Network [1]

In object detection neural networks, multiple layers are used to extract features. The layers are then stacked up for input to be fed through. Error comparison is done at every layer. The "lower layers" (close to the input) define the basic features like object boundaries (left curved side of object for example). The "upper layers" combine these boundaries (general shape of object is discovered) [1].

For object detection and other ML tasks, there are two broad categories: Classification and Regression. For classification, an input is mapped to a class value. An example is email being classified as spam or not spam. Regression seeks to map an input to a continuous range of values. An example is inputting data about a house and predicting its monetary value in dollars. For this project I trained both classifiers and regression models. Classifiers will classify the entire image if it is trash or not. Regressors will predict the bounding box(pixel coordinates) of the object in the image.

<u>Experiments to Decide Final Model Type:</u>

A variety of different model types were experimented with to determine the final one. In Design 1, Haar-Cascade, Detecto, and YoloV5 models were trained. Detecto was found to be incompatible, while Haar-Cascade could not perform well for our applications. The YoloV5 architecture may be used as a backup model in case my custom regressor model is not robust enough.

In Design 2, SVM models were experimented with. They are easy to train, but proved infeasible because the model sizes are too large (4.5 Gb for regressor). The Tensorflow framework for training custom neural networks proved to be an immediate improvement over SVM models. However, building a regressor ML model from scratch proved to be extremely difficult: robustness was always lacking for our desired task. As a result, a Yolo-NAS regressor model was trained. While I cannot claim to have built the underlying architecture behind Yolo-Nas, I understand and appreciate its complexity.

ROS:

The main framework that is used to take all incoming data from sensors, manipulate and analyze those signals, and output the correct commands to our DC and servo motors is called the Robot Operating System (ROS). In our case, since the project would integrate better with the second iteration of ROS called ROS2, we chose this to install onto the host PC. The architecture of the framework that is implemented is gone into detail in Figure 4 below. Most input sensors (cameras, LIDAR, IMU's, etc.) have pre-coded drivers that integrate well with ROS2 and can make data communication extremely efficient.

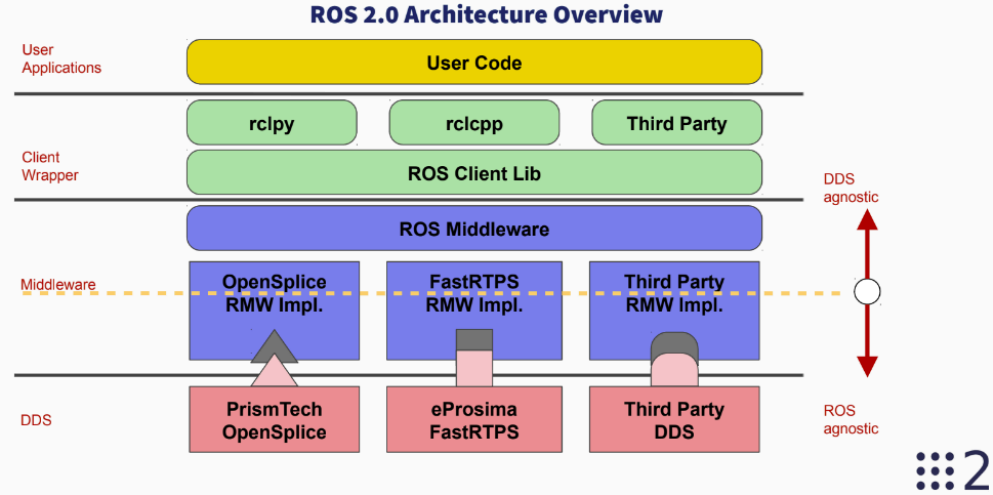

Figure 4: ROS2 Foxy Package internal implementation

For the ROS2 framework, in our system there were a few background projects that were looked at but for most ROS projects all frameworks are vastly different depending on sensors and control algorithms. When coding the Teensy 4.1 microcontroller, the computer needs to connect and read and write ROS topics. This was a main problem when first setting up our robot. Micro_ROS connects the ROS topics being sent and received from both parties. The connection is shown below in Figure 4.

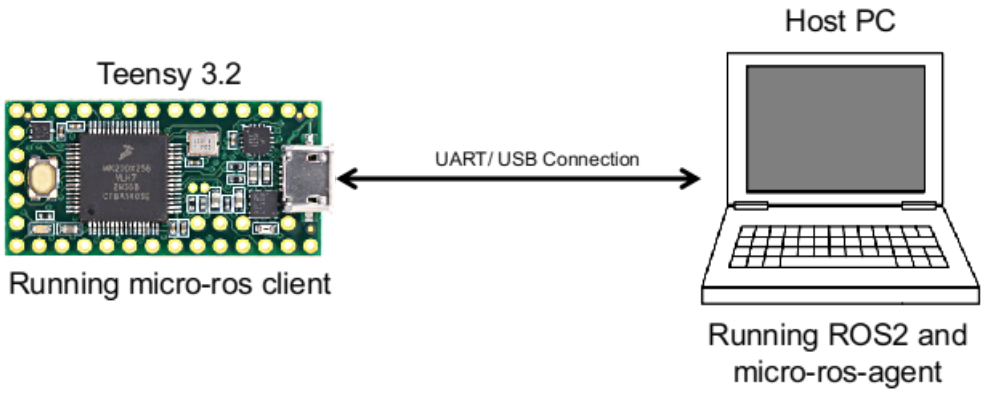

Figure 5: The Micro_ROS connection between the host and the Teensy 4.1 microcontroller client

Timeline

Throughout this project, we utilized Agile project management methods and iterative development. At the start of Design 1, we decided to separate people into areas of work, Nicholai worked on machine learning, James on ROS, Jorge on PCB and overall help with all teammates, Michele and Natalie on Physical build. Nearing the end of the final build, our team was flexible with some jumping around trying to help people finish their portion of the work. We organized work depending on who was working on what and specific dependencies related to each section of the project. Sometimes work had to be halted due to these dependencies so finding them early and planning around them was crucial to avoid others waiting. We set short term and long term goals for the project and re-evaluated each deliverable to keep the timeline in focus for both semesters. Our team met almost everyday in person at the lab and we had bigger re-evaluation and planning sessions after each deliverable was due to plan for the next.

Pre-Alpha (2 week)

Physical

- First round of hardware bought
- Planned our ideas of what the physical model would look like
  - Use a RC chassis with tracks as a base
  - Pin-ball arms to pick up the trash
  - A bucket that would scoop the trash
  - A bin to discard the trash

ML

- Trained placeholder ML model using the “Detecto” library. It was found to be incompatible.

ROS

- Setting up the ROS2 framework and Arduino IDE on the UP Squared board computer.

Design Prototype (2 weeks)

Physical

- Cardboard design complete
  - The conceptual idea of what we had was made using cheap material

ML

- Trained Haar-Cascade model. This technique did not create robust models for our application.
- Trained Yolo-V5 model. It was discovered to be very robust.

ROS

- Setting up the ROS2 framework and Arduino IDE while researching SLAM and NAV2.
- Coding the OpenCV and ROS2 compatible node that manipulates the input image frame stream.

Alpha (2 weeks)

Physical

- Changed to brush method
  - Designed a more reliable mechanism to pick up the trash

ML

- Trained initial SVM classifier model.

ROS

- Researching ROS2 and SLAM
- Getting the first machine learning model integrated into the image subscriber ROS2 node.
- Beginning code of the keyboard press first model

Beta (3 weeks)

Physical

- Final Chassis used
  - Needed a new base that would fit the scale of the project
- Redesigned the size and connection between all the pieces with the new chassis

ML

- Additional training images were taken: improved model accuracy.
- SVM classifier model was improved and the first regression SVM model was trained.
- SVM models were discovered to be insufficient for our application
- First MLP/CNN model was trained on Tensorflow.
- Alpha tests performed to compare SVM and MLP/CNN type models.

ROS

- Reinstalling all ROS2, Arduino, SLAM, NAV2, OpenCV software for our new laptop that replaced the on board computer of the UP Squared.
- Finished the keyboard press code for remote manipulation of the robot.

Release Candidate (2 weeks)

Physical

- PCB finalized
- Redesigned bucket to be detachable

ML

- Training photos taken with the Intel Real-Sense camera: improved model.
- Beta Testing done on MLP/CNN model
- Additional training photos were captured

ROS

- Researching SLAM and NAV2 and doing odometry calculations on our tank tread robot.
- Attempting to integrate our robot with SLAM and NAV2

Production Release (3 weeks)

Physical

- Finished 3D printing all the parts
  - Bucket
  - Trash bin
  - Laptop holder
  - PCB/battery/Lidar case
- Wire management

ML

- Trained first Regressor MLP/CNN model.
- Used results from Beta testing to improve the model
- My overall ML architecture was discovered to be too simple for our application.
- Yolo-NAS model was trained and integrated.

ROS

- Finishing the new autonomous robot code. This included editing the image analyzer node and a whole new Arduino code file. This was the final touch for the ROS2 framework.

Design

ROS:

The ROS2 framework connects all other peripherals together and its design is imperative to the robot working correctly. The first streams of data that are being inputted into the framework are the sensor data and keyboard presses. The camera is publishing the "/camera/image_raw" and other topics for the IMU data. The LIDAR is publishing the "/scan" topic that is a depiction of an environment across a horizontal plane. These incoming topics are sent to three different locations all having their own analyzation techniques and algorithms.

The camera topics that are in the framework go into a python script that is integrated with the ROS, Super-Gradients(YOLO-NAS), and OpenCV libraries. This script analyzes each frame that the camera topic sends by running it through the Yolo-Nas machine learning model. OpenCV is used to manipulate the image into an acceptable format to run through YoloNas. When a lighter is in frame (up to 180 cm from the camera), the ML model outputs the pixel coordinates of a bounding box surrounding the trash (lighter). The location and area of the bounding box affects how the robot moves. If the bounding box is off center the robot will automatically correct itself so the bounding box is always in the middle of the frame. When the robot gets close enough based on the area of the box the robot goes into pick up mode and picks up the trash automatically. The robot then goes back into searching mode and the data stream loop starts again.

The LIDAR “/scan” topic is the main peripheral that is used in the autonomous navigation of our robot which is in progress. The SLAM ROS2 package takes in this scan topic and the odometry data of the robot and outputs a map of the environment with all obstacles included. This map topic then would have gotten sent to the NAV2 ROS2 package with other robot state publishing topics that would make it possible to navigate to any point in space digitally. Since the NAV2 and SLAM topics are not implemented on our robot after extended research and effort, the LIDAR ROS2 driver is running only on our robot for this peripheral.

The data streams of either of a 0, 1, 2, 3, 4, or 5 is sent to the micro_ROS agent and then to the Arduino code subscribing to it. A 0 means the robot is in searching mode. This is when the robot is spinning in a circle trying to find any objects that match our machine learning model. A 1 means the robot needs to correct left. A 2 means the robot needs to correct right. A 3 means that the robot is far away from the object and the bounding box is in the center. The robot will move forward at an increased speed since the object is far away. A 4 means that the distance and location is correct and all the robot needs to do is lower the bucket and sweep the trash. A 5 is when the robot is close to the object but still not at the required distance. The robot then has a 100ms burst of speed until the object is close enough.

Physical Build:

Any connections aside from the DC motors, battery, servo motors and the micro-usb connection from the processor to the Teensy were made to be internal connections on a PCB. This was done to make it easier to perform tests as well as make it a lot less likely that connections would be severed in the middle of testing. As previously mentioned, the communication protocol used between the Teensy and the Adafruit Featherwing was I2C, which was also made as an internal connection on the PCB.

The robot’s physical design includes a bucket with a sweeper. The trash is swept in with a servo motor. The bucket then rotates backwards with the use of another servo motor, where the trash will fall into a trash bin. This trash bin slides in and out for ease of access. The robot also has encoder motors that can be used for autonomous mapping by increasing the consistency in the robot’s speed calculations. On the bottom of the robot is the camera for the best visual positioning to see the trash on the ground. On top of the robot is the rest of the components including the battery, laptop, PCB, and lidar. The lidar sits on top of a compartment for the best view of the environment. The PCB and battery sit encased in the compartment for a secure build. All external parts of the build were 3D printed for better durability.

Tools and Standards

ML:

To manage python dependencies, Anaconda was used. Anaconda acts as a virtual environment for python libraries to eliminate dependency conflicts when too many libraries are installed together. Also, a suite of python IDEs can be installed inside of Anaconda [2].

The Tensorflow python library was used to create neural network architectures. It has a vast API to create a multitude of architectures with many options. Designing, creating and loading ML can be completed using Tensorflow. Other notable libraries include Scikit-Learn and opencv [3].

The final model (used for production release) was created using the Super-Gradients API. Super-Gradients allows for abstractions and ease to train Yolo-NAS type ML models.

Yolo-NAS itself is an extremely robust neural network architecture built by ML scientists. Its main purpose is object detection (drawing bounding boxes). Yolo-NAS itself is built upon the Py-Torch library [6].

3D printing:

Our team used Solidworks to 3D model all of the components that were originally created out of cardboard and wood. We then used UltiMaker Cura software slicing software to prepare our print.

PCB:

For developing the PCB, the electronic design automation software called EAGLE was used due to several of the components having footprints available on there, which helped to facilitate the design process by a large margin. Another reason why it was chosen was due to it having forward annotation, which is the process of sending schematic changes to the actual PCB layout, which helps to save time, especially when making revisions to an already existing PCB layout.

Need and Impact

The main benefit of LEP is the environmental impact it can make by cleaning up areas, especially ones where part of the attraction is the nature present, such as parks. LEP will also almost always be actively searching for trash to pick up aside from when it needs to charge, which means that there can always be a “worker” present that will be picking up trash. The main cultural impact of LEP is that it promotes a culture that embraces technological advances and discoveries since it is a robot that will be seen by people almost everyday, which would help get them used to being in the presence of robots. By having LEP be consistently in the presence of others, it will help make people feel more comfortable when around robots by giving them a positive experience. This is especially the case with people who are likely to have a poor understanding of robots and AI, such as young children. The main social impact of LEP is that it can help increase the trust that the public has on robots and AI since there are still several people who are distrusting of them and seeing one provide such a direct positive impact on the environment might allow people to see the potential of using robots and AI in everyday life. An example of this mistrust can be seen in how hesitant people are to have a car that has self-driving features due to believing that it would be likely for the car to make a mistake and crash. The one potential economic negative impact of LEP is that it can replace human workers that were responsible for picking up the trash in parks, which can lead to an increase in unemployment. However, this can be mitigated by creating job positions that are responsible for supervising LEP since it can potentially make mistakes or get stuck. It would not be hard to at least have some of the workers that were originally responsible for keeping the parks clean be trained to supervise LEP. They would be taught the behaviors of LEP and what it is expected to do so they can keep a lookout for any anomalous behaviors. Their previous experience in keeping the parks clean will help them recognize when LEP is about done with an area. In order to facilitate this supervision, they could both be provided with a live feed of the camera view for each individual LEP unit and their live GPS coordinates. LEP can also be used as a basis for other projects that involve autonomous navigation and the ability to retrieve certain objects once it identifies them. For example, a project could focus on extending LEP’s capabilities by having it be able to recognize when there is large amounts of dust present and adding a retrieval method that allows it to pick up the dust, such as a vacuum.

Results

Testing

ML Tests

The two most important tests are latency and performance. Latency measures the time between frame received and predict() called. This was completed using the simple stopwatch timer module from the python library: Time. Performance was first tested using a webcam and then later with the Intel Real-Sense integrated with the robot. Trash objects (usually the lighter object for consistency) were placed at 3 distances. At these distances, the lighter orientation was changed in 5 ways making a total of 15 total tests. Additionally, the false positive rate was repeated for these 15 tests: verify that the model does not detect an empty environment as trash at the 3 distances. The memory usage test was not attempted because the performance test gives enough information itself.

Latency tests

SVM style classifier model	300-400 ms delay on laptop
MLP/CNN neural network classifier mode	20-30 ms on laptop
MLP/CNN neural network regressor model	60-80 ms on laptop
Yolo-NAS	80-100 ms on laptop

Performance tests:

Model Type	Close (20 cm)	Medium (60 cm)	Far (100 cm)
SVM classifier	5/5	3/5	0/5
MLP/CNN classifier	3/5	2/5	0/5
MLP/CNN regressor	2/5	0/5	0/5
Yolo-NAS	3/5	5/5	5/5

Note: 5/5 means 5 out of 5 correct classification/ bounding box drawings at that distance

False Positive Tests:

Model Type	Close (20 cm)	Medium (60 cm)	Far (100 cm)
SVM classifier	5/5	5/5	3/5
MLP/CNN classifier	2/5	1/5	0/5
MLP/CNN regressor	2/5	0/5	0/5
Yolo-NAS	5/5	4/5	5/5

Note: 5/5 means 5 out of 5 correct behavior, meaning no false positives

Physical build Tests

From the Alpha test plan, motor functions and some special cases were tested for robustness. We also tested the average power draw from the battery where we found the robot would run for 2-3 hours. From the Beta, we have tested the camera position which creates much more consistent and accurate results from the ML model detecting trash. Previously, there were issues with the camera's position as part of the robot would be in the picture which would throw off the ML model. The main test that was done on the PCB was confirming that the power supply pins were getting the expected voltage values. The test simply consisted of connecting the PCB to the 11.1V battery after everything was successfully soldered onto it and using a digital multimeter to confirm the values (place the ground probe on any ground pin on the PCB and the VCC probe on the pin you are interested in getting a voltage value from). The main concern that led to performing this test was the buck converter, which is used to lower the 11.1V from the battery to 5V, which is used to power the servo motors, the encoders for the DC motor and the high voltage pin for the two level shifters. When we first started testing, any pins that were connected directly from the positive terminal of the battery were receiving the expected 11.1V and any pins that needed 3.3V from the Teensy was receiving the expected 3.3V, but any pins that were supposed to receive the 5V from the buck converter were not receiving anything. After having three PCBs and three buck converters be lost due to severed routes, heat damage etc., we finally were able to treat a PCB with enough care that it was outputting 5V from the buck converter as expected. Other tests outlined in the Beta plan were unable to be completed due to more electronic failure complications. An UpSquared, RasPi, featherwing motor controller board, and a PCB got destroyed during testing of the robot. Something we would change from trying to test our robot is trying to put protective measures in place to avoid the short circuiting of circuits.

ROS Tests

The ROS tests for our robot were and are very simple and consist of connecting everything successfully. The alpha tests were to make sure that everything is receiving and sending data and not worrying about if it was correct. The vertical slice of the alpha made it so we can focus on making the framework correct and not the data being sent through it. The beta test was much more focused on the data. The tests made were to ensure that the Teensy 4.1 microcontroller was receiving and sending the correct data so it can control the motors with efficiency. This was based on the key presses from our host computer. The tests that were implemented in this milestone were only bug fix tests. This checking demonstrates that if we have a problem elsewhere in another part of the robot we know it is not the code or the framework in ROS2.

Final State

ML:

The Yolo-Nas model outperformed all the custom models I created myself (SVM and MLP/CNN types). The YoloNas architecture had more complexity and nuance than the architectures I created myself. Also, the Yolo framework has a strong built-in image augmentation strategy that greatly contributes to its overall accuracy in many environments. Therefore, I chose Yolo-Nas as my final model to be used.

The number of classes known to the ML model is its largest drawback. The Yolo-Nas architecture needed to be quickly created before the final deadline because my custom models were not robust enough. As a result, only the "empty floor" and "lighter" class could be trained

with it. If more time was available, other trash objects like the bottle cap or paper could also be added.

Also, since our programs are running on a small laptop, the performance is better than most single-board computers. The Yolo-Nas model would need to be further fine-tuned and its file size shrunk in order to perform well on a Raspberry PI for example. Other things could be done to improve performance as well: convert python ML code to C++, and (or) utilize an AI specific chip on a single-board computer like the Beagele-Bone-AI.

Physical Build:

We ended up using revision #2 for the PCB rather than create a new revision for the production release. This was because it was not really necessary to modify the current PCB in the end due to us simply using a laptop as our processor rather than a board, such as the UP Squared, that has male headers that can be used to connect the PCB. The laptop also has its own battery supply, which means that the power jack is no longer needed since it was meant to power our processor. This means that it is not really necessary either to thicken up the ground routes on the PCB since it will no longer be carrying as high of a current. The main challenge that we faced when it came to the PCB was that it was very easy to cause a short circuit or damage the PCB itself. This was mainly addressed by using the proper tools that help confirm that PCB was behaving as expected, such as by using the oscilloscope to confirm that the correct voltage values are getting to where they need to be.

For the physical build of our robot, we're using a larger and more reliable chassis than we had originally. A challenge we had with the old chassis was with the weight unbalance, which threw the whole robot off the ground. All the parts are now 3D printed for a more robust build. It was challenging using cardboard and any cheap material because it would wear and tear as we kept using it, making it unreliable.The new removable design of the bucket allows for future adjustments and makes it easily replaceable. The trash bin was also designed so that it can be removed completely for effortless trash removal. The brush method is also used to lesson the amount of servo motors and it proved to be more reliable than the method we used previously. We also ran into some issues of short circuiting the UP Squared, as a result we used a laptop in replacement. With the laptop in use we had to account for the placement of it. Therefore we created a laptop holder that would keep it in place on top of our robot. We also now have a case that holds the PCB, battery, and Lidar for a cleaner and secure build.

Even though the chassis we are currently using is reliable for the time and monetary restriction given, in the future a larger and better quality chassis would be used. We would also try a different method of picking up the trash. The current sweeping method will sometimes get the trash caught on the edge of the bucket. In the future we also wanted to create a doc the robot would go back to when the trash bin was filled or if it needed to be charged.

ROS:

The final state of the ROS2 framework consists of an automated data stream that needs no outside interaction. The camera and LIDAR send ROS2 signals to the framework and the output is the motor commands. The whole ROS2 framework with all of the custom nodes used is outlined in the figure below.

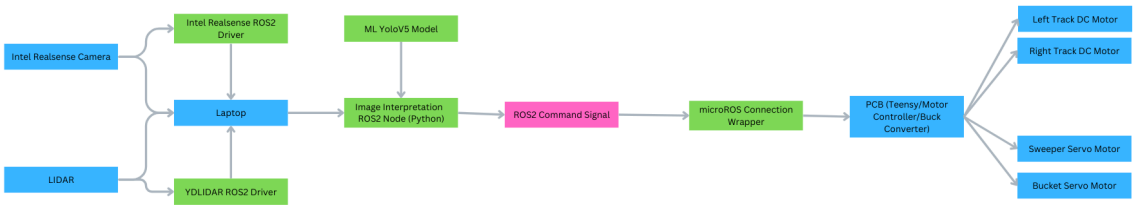

Figure 6: ROS2 Final Framework

The main challenge when working with ROS2 is trying to configure your own custom framework based on your unique needs. The coding portion of the different nodes surprisingly took the least amount of time. The installation of all the different dependencies, drivers, and the basic framework design took the bulk of the time when developing our robot. Having documentation on certain aspects of ROS2 development would expedite many setbacks that we encountered. Since we were using a custom ROS2 framework that was largely never done before, documentation could be extremely helpful for other teams in the future. The SLAM and NAV2 package implementation of fully autonomous navigation was not completed, however, a rough framework was completed. This framework was never implemented and we decided to take the meaning autonomous into another direction. The robot is not affected by this in our final build and can drive fully autonomously without the use of the two pre-built ROS2 packages.

Conclusion

This project has the potential for strong environmental and economic impact on society as its ability to pick up trash creates a cleaner and healthier environment for those around it. In general, robotic automations should be researched and studied as they have potential for possessive change for society. Our goal for this project was to learn more about these robotic solutions and try to solve one for ourselves. Currently, our robot is running with keyboard commands with the LiDar and camera giving spatial data to ROS for the ML model to interface. With how many issues we have had this semester with lots of unfortunate electronic explosions, the main goal of making the robot completely autonomous is unrealistic. We still want to interface with SLAM to map the environment for cleaning and to have the robot detect trash but the logic for picking up trash on its own is becoming a stretch goal. To conclude, there were many hardships throughout this semester, but lots of lessons learned.

References

[1] M. I. Jordan and T. M. Mitchell, “Machine learning: Trends, Perspectives, and prospects | science,” Machine learning: Trends, perspectives, and prospects, https://www.science.org/doi/10.1126/science.aaa8415 (accessed Nov. 3, 2023).

[2] “About anaconda,” Anaconda, https://www.anaconda.com/about-us (accessed Nov. 3, 2023).

[3] “Tensorflow,” TensorFlow, https://www.tensorflow.org/ (accessed Nov. 3, 2023).

[4] J. Bai, S. Lian, Z. Liu, K. Wang, and D. Liu, “Deep learning based robot for automatically picking up garbage on the grass,” IEEE Transactions on Consumer Electronics, vol. 64, no. 3, pp. 382–389, 2018.

[5] ROBOTECH, “Dustclean,” *ROBOTECH srl*. [Online]. Available: https://www.robotechsrl.com/dustclean-en-robot-sweeper/. [Accessed: 20-Feb-2023].

[6] “Supergradients is an open source neural network training library for pytorched based computer vision models.,” SuperGradients, https://www.supergradients.com/ (accessed Nov. 27, 2023).